Parallel Matlab: The Next Generation

Jeremy Kepner* (kepner@ll.mit.edu) and Nadya Travinin (nt@ll.mit.edu)
MIT Lincoln Laboratory, Lexington, MA 02420

Abstract

The true costs of high performance computing are currently dominated by software. Addressing these costs requires shifting to high productivity languages such as Matlab. The development of MatlabMPI (www.ll.mit.edu/MatlabMPI) was an important first step that has brought parallel messaging capabilities to the Matlab environment, and is now widely used in the community. The ultimate goal is to move beyond basic messaging (and its inherent programming complexity) towards higher level parallel data structures and functions. The pMatlab Parallel Toolbox provides these capabilities, and allows any Matlab user to parallelize their program by simply changing a few characters in their program. The performance has been tested on both shared and distributed memory parallel computers (e.g. Sun, SGI, HP, IBM, Linux and MacOSX) on a variety of applications.

1 Introduction

MATLAB®[1] is the dominant interpreted programming language for implementing numerical computations and is widely used for algorithm development, simulation, data reduction, testing and system evaluation. The popularity of Matlab is driven by the high productivity that is achieved by users because one line of Matlab code can typically replace ten lines of C or Fortran code. Many Matlab programs can benefit from faster execution on a parallel computer, but achieving this goal has been a significant challenge (see [2] for a reveiw). MatlabMPI [3, 4, 5] has brought parallel messaging capabilities to hundreds of Matlab users and is being installed in several HPC centers.

The ultimate goal is to move beyond basic messaging (and its inherent programming complexity) towards higher level parallel data structures and functions. pMatlab achieves this by combinng operator overloading (first demonstrated in Matlab*P) with parallel maps (first demonstrated in Lincoln's Parallel Vector Library - PVL) to provide implicit data parallelism and task parallelism. In addition, pMatlab is built on top of MatlabMPI and is a "pure" Matlab implementation which runs anywhere Matlab runs, and on any heterogeneous combination of computers. pMatlab allows a Matlab user to parallelize their program by changing a few lines. For example, the following program is a parallel implementation of a classic "corner turn" type of calculation commonly used in signal processing

```
pMATLAB_Init; Ncpus=comm_vars.comm_size;     % Initialize
mapX = map([1 Ncpus/2],{},[1:Ncpus/2])       % Map X
mapY = map([Ncpus/2 1],{},[Ncpus/2+1:Ncpus]) % Map Y
X = complex(rand(N,M,mapX),rand(N,M,mapX));  % Create X
Y = complex(zeros(N,M,mapY);                 % Create Y
coefs = ...              % Local matrix of coefs.
weights = ...            % Local matrix of weights.
Y(:,:) = conv2(coefs,X); % Parallel filter + corner turn.
Y(:,:) = weights*Y;      % Parallel matrix multiply.
pMATLAB_Finalize; exit;  % Finalize pMATLAB and exit.
```

The above example illustrates several powerful features of pMatlab: independence of computation and parallel mapping, "automatic" parallel computation, and data redistribution via operator overloading.

2 Performance Results

The vast majority of potential Matlab applications are "embarrassingly" parallel and require minimal performance out of the communication capabilities in pMatlab. These applications exploit coarse grain parallelism and communicate rarely. Figure 1 shows the speedup

*This work is sponsored by the High Performance Computing Modernization Office, under Air Force Contract F19628-00-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government

[1]MATLAB is a registered trademark of The Mathworks, Inc.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **20 AUG 2004**	2. REPORT TYPE **N/A**	3. DATES COVERED **-**

4. TITLE AND SUBTITLE **Parallel Matlab: The Next Generation**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **MIT Lincoln Laboratory, Lexington, MA 02420**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM001694, HPEC-6-Vol 1 ESC-TR-2003-081; High Performance Embedded Computing (HPEC) Workshop (7th)., The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**UU**	**34**	

obtained on a typical parallel clutter simulation. Never-the-less, measuring the communication performance is useful for determining which applications are most suitable for pMatlab. pMatlab has been run on several platforms. It has been benchmarked and compared to the performance of the underlying MatlabMPI upon which it is built. These results indicate that the overhead of pMatlab is minimal (see Figure 2), the primary difference is in the latency: 70 milliseconds for pMatlab compared to 35 milliseconds for MatlabMPI. Both pMatlab and MatlabMPI match the performance of native C MPI [1] for very large messages.

These results indicate that it is possible to write effective parallel programs in Matlab with minimal modifications to the serial Matlab code. In addition, these capabilities can be provided in a library that is written entirely in Matlab. Ultimately, it is our goal to establish a unified interface for parallel Matlab that a broad community supports. We are actively collaborating with Ohio State, UC Santa Barbara and the MIT Laboratory for Computer Science to provide a single Unified Parallel Matlab interface that is supported by multiple underlying implementations (e.g. pMatlab and Matlab*P).

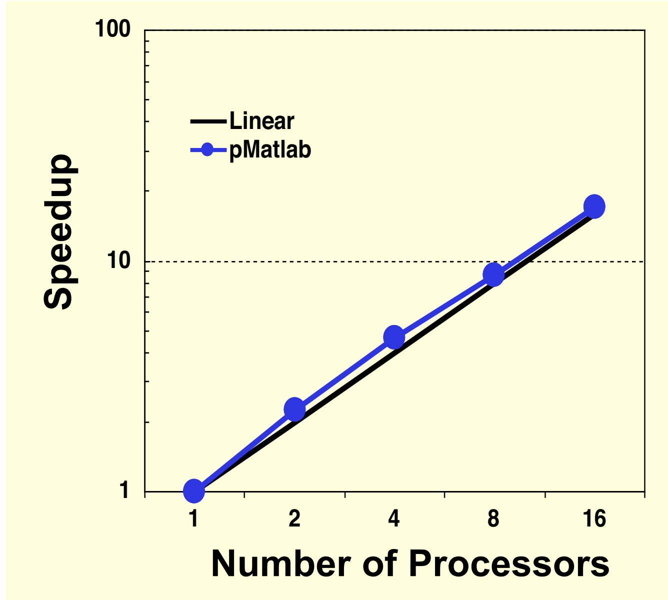


Figure 1: **Clutter Simulation Speedup.** Parallel performance speedup of a radar clutter simulation on a cluster of workstations.

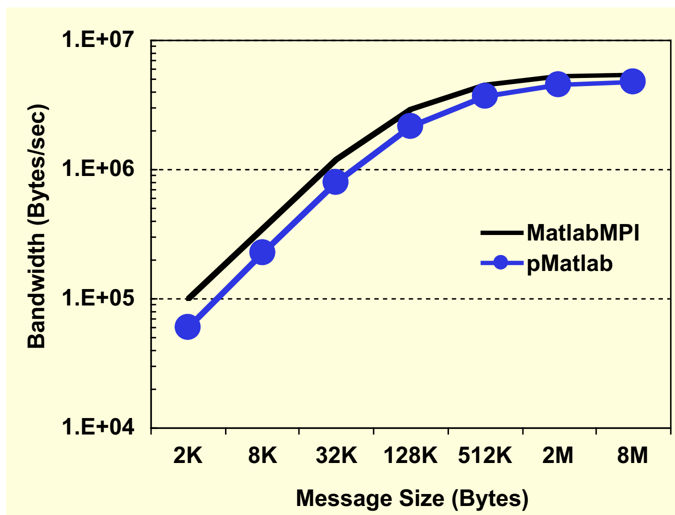


Figure 2: **pMatlab vs. MatlabMPI Bandwidth.** Communication performance on a “Ping Pong” benchmark as a function of message size on a Linux cluster. pMatlab equals underlying MatlabMPI performance at large message sizes. Primary difference is latency (70 vs. 35 milliseconds).

References

[1] Message Passing Interface (MPI), http://www.mpi-forum.org/

[2] R. Choy, Parallel matlab survey, www.mit.edu/∼cly/survey.html

[3] J. Kepner, Parallel Programming with MatlabMPI, HPEC 2001 Workshop

[4] J. Kepner, 300x Matlab, HPEC 2002 Workshop

[5] J. Kepner and S. Ahalt MatlabMPI, submitted to the Journal of Parallel and Distributed Computing, www.arxiv.org/abs/astro-ph/0305090

Abstract

Presentation

Back to Agenda

Next Session

Parallel Matlab: The Next Generation

Dr. Jeremy Kepner /MIT Lincoln Laboratory

Ms. Nadya Travinin / MIT Lincoln Laboratory

This work is sponsored by the Department of Defense under Air Force Contract F19628-00-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the author and are not necessarily endorsed by the United States Government.

Outline

- **Introduction**
  - ***Motivation***
  - ***Challenges***
- **Approach**
- **Performance Results**
- **Future Work and Summary**

Motivation: DoD Need

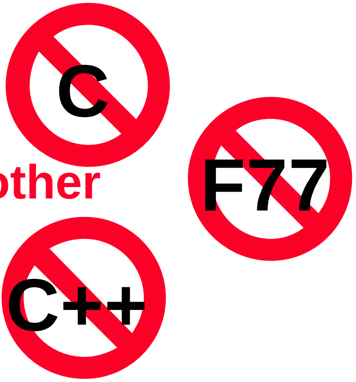
- **Cost**

= 4 lines of DoD code

- **DoD has a clear need to rapidly develop, test and deploy new techniques for analyzing sensor data**
  - **Most DoD algorithm development and simulations are done in Matlab**
  - **Sensor analysis systems are implemented in other languages**
  - **Transformation involves years of software development, testing and system integration**

- **MatlabMPI allows any Matlab program to become a high performance parallel program**

Challenges: Why Has This Been Hard?

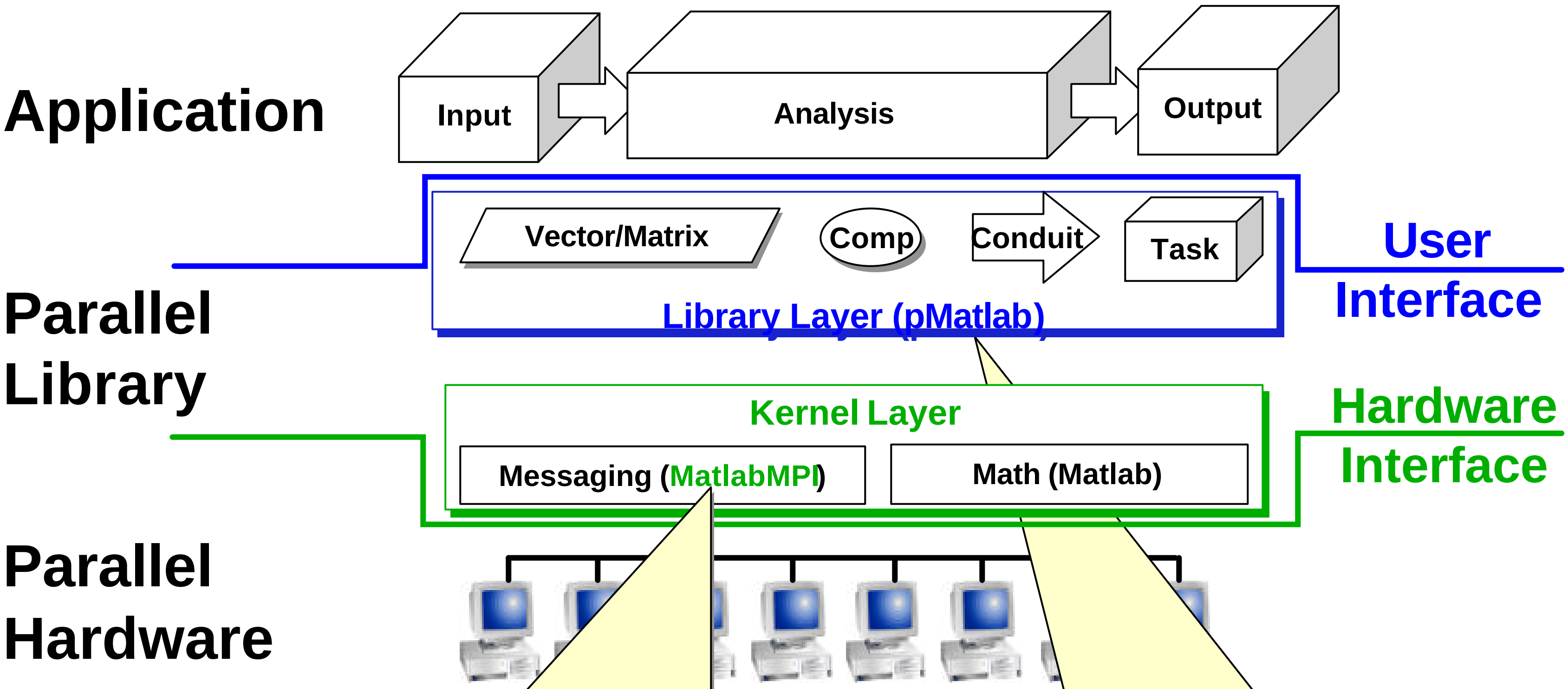


- **Productivity**
  - **Most users will not touch any solution that requires other languages (even cmex)**
- **Portability**
  - **Most users will not use a solution that could potentially make their code non-portable in the future**
- **Performance**
  - **Most users want to do very simple parallelism**
  - **Most programs have long latencies (do not require low latency solutions)**

Outline

- Introduction
- **Approach**
  - ***MatlabMPI messaging***
  - ***pMatlab programming***
- Performance Results
- Future Work and Summary

MatlabMPI & pMatlab Software Layers



- Can build a parallel library with a few messaging primitives
- MatlabMPI provides this messaging capability:

  MPI_Send(dest,comm,tag,X);
  X = MPI_Recv(source,comm,tag);

- Can build a application with a few parallel structures and functions
- pMatlab provides parallel arrays and functions

  X = ones(n,mapX);
  Y = zeros(n,mapY);
  Y(:,:) = fft(X);

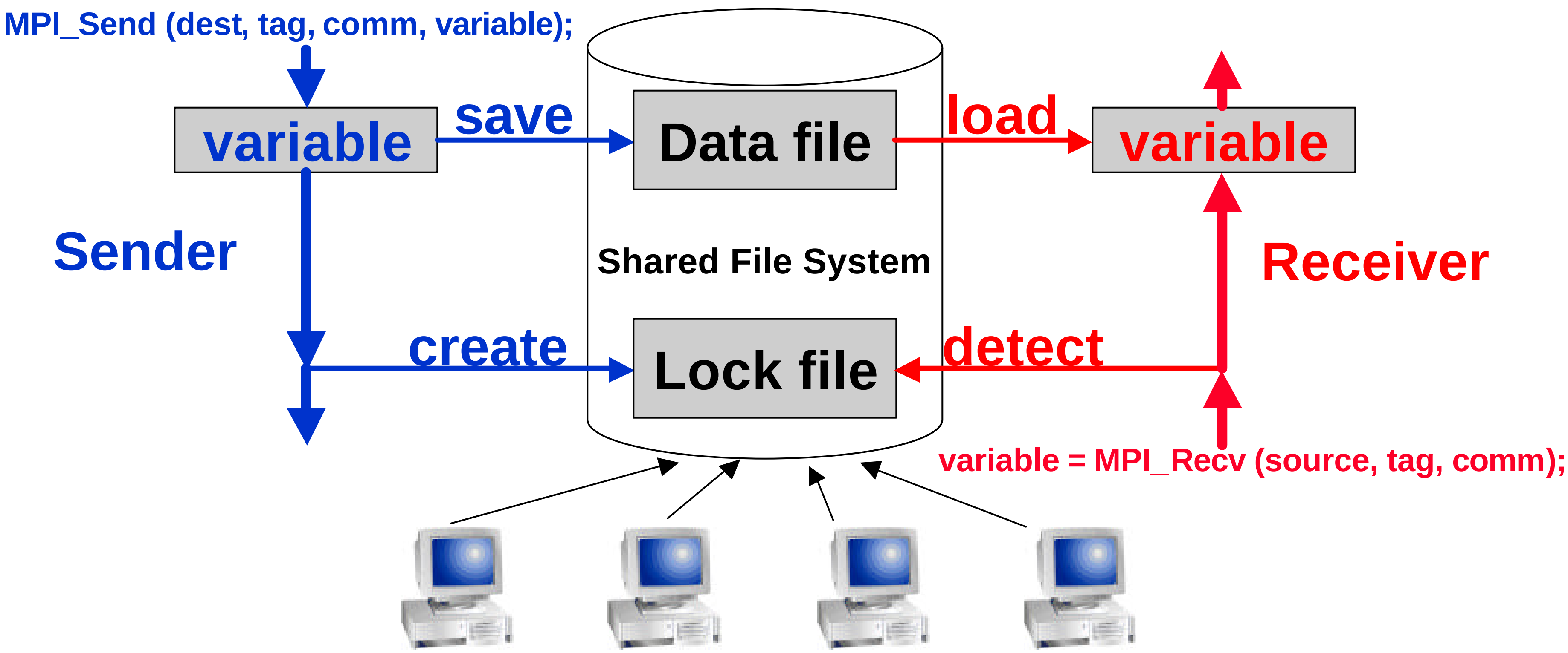
MatlabMPI fuctionality

- “Core Lite” Parallel computing requires eight capabilities
  - MPI_Run launches a Matlab script on multiple processors
  - MPI_Comm_size returns the number of processors
  - MPI_Comm_rank returns the id of each processor
  - MPI_Send sends Matlab variable(s) to another processor
  - MPI_Recv receives Matlab variable(s) from another processor
  - MPI_Init called at beginning of program
  - MPI_Finalize called at end of program

- Additional convenience functions
  - MPI_Abort kills all jobs
  - MPI_Bcast broadcasts a message
  - MPI_Probe returns a list of all incoming messages
  - MPI_cc passes program through Matlab compiler
  - MatMPI_Delete_all cleans up all files after a run
  - MatMPI_Save_messages toggles deletion of messages
  - MatMPI_Comm_settings user can set MatlabMPI internals

MatlabMPI: Point-to-point Communication

- Any messaging system can be implemented using file I/O
- File I/O provided by Matlab via load and save functions
  - Takes care of complicated buffer packing/unpacking problem
  - Allows basic functions to be implemented in ~250 lines of Matlab code



- Sender saves variable in Data file, then creates Lock file
- Receiver detects Lock file, then loads Data file

Example: Basic Send and Receive

- **Initialize**
- **Get processor ranks**

- **Execute send**
- **Execute recieve**

- **Finalize**
- **Exit**

```
MPI_Init;                            % Initialize MPI.
comm = MPI_COMM_WORLD;               % Create communicator.
comm_size = MPI_Comm_size(comm);     % Get size.
my_rank = MPI_Comm_rank(comm);       % Get rank.
source = 0;                          % Set source.
dest = 1;                            % Set destination.
tag = 1;                             % Set message tag.

if(comm_size == 2)                   % Check size.
 if (my_rank == source)              % If source.
  data = 1:10;                       % Create data.
  MPI_Send(dest,tag,comm,data);      % Send data.
 end
 if (my_rank == dest)                % If destination.
  data=MPI_Recv(source,tag,comm);    % Receive data.
 end
end

MPI_Finalize;                        % Finalize Matlab MPI.
exit;                                % Exit Matlab
```

- **Uses standard message passing techniques**
- **Will run anywhere Matlab runs**
- **Only requires a common file system**

pMatlab Goals

- *Allow a Matlab user to write parallel programs with the least possible modification to their existing matlab programs*

- **New parallel concepts should be intuitive to matlab users**
  - **parallel matrices and functions instead of message passing**
  - **Matlab*P interface**

- **Support the types of parallelism we see in our applications**
  - **data parallelism (distributed matrices)**
  - **task parallelism (distributed functions)**
  - **pipeline parallelism (conduits)**

- **Provide a single API that potentially a wide number of organizations could implement (e.g. Mathworks or others)**
  - **unified syntax on all platforms**

- **Provide a unified API that can be implemented in multiple ways,**
  - **Matlab*P implementation**
  - **Multimatlab**
  - **matlab-all-the-way-down implementation**
  - **unified hybrid implementation (desired)**

Structure of pMatlab Programs

Initialize globals

```
pMATLAB_Init;

mapX = map([1 N/2],{},[1:N/2]);
mapY = map([N/2 1],{},[N/2+1:N]);
X = ones(n, mapX);
Y = zeros(n, mapY);
Y(:,:) = fft(X);

pMATLAB_Finalize;
```

Clear globals

Map to sets of processors

Distributed matices

Parallel FFT and "Corner Turn" Redistribution

- Can parallelize code by changing a few lines
- Built on top of MatlabMPI (pure Matlab)
- Moving towards Matlab*P interface

pMatlab Library Functionality

- "Core Lite" Provides distributed array storage class (up to 4D)
  - Supports reference and assignment on a variety of distributions:

    Block, Cyclic, Block-Cyclic, Block-Overlap

  Status: Available

- "Core" Overloads most array math functions
  - good parallel implementations for certain mappings

  Status: In Development

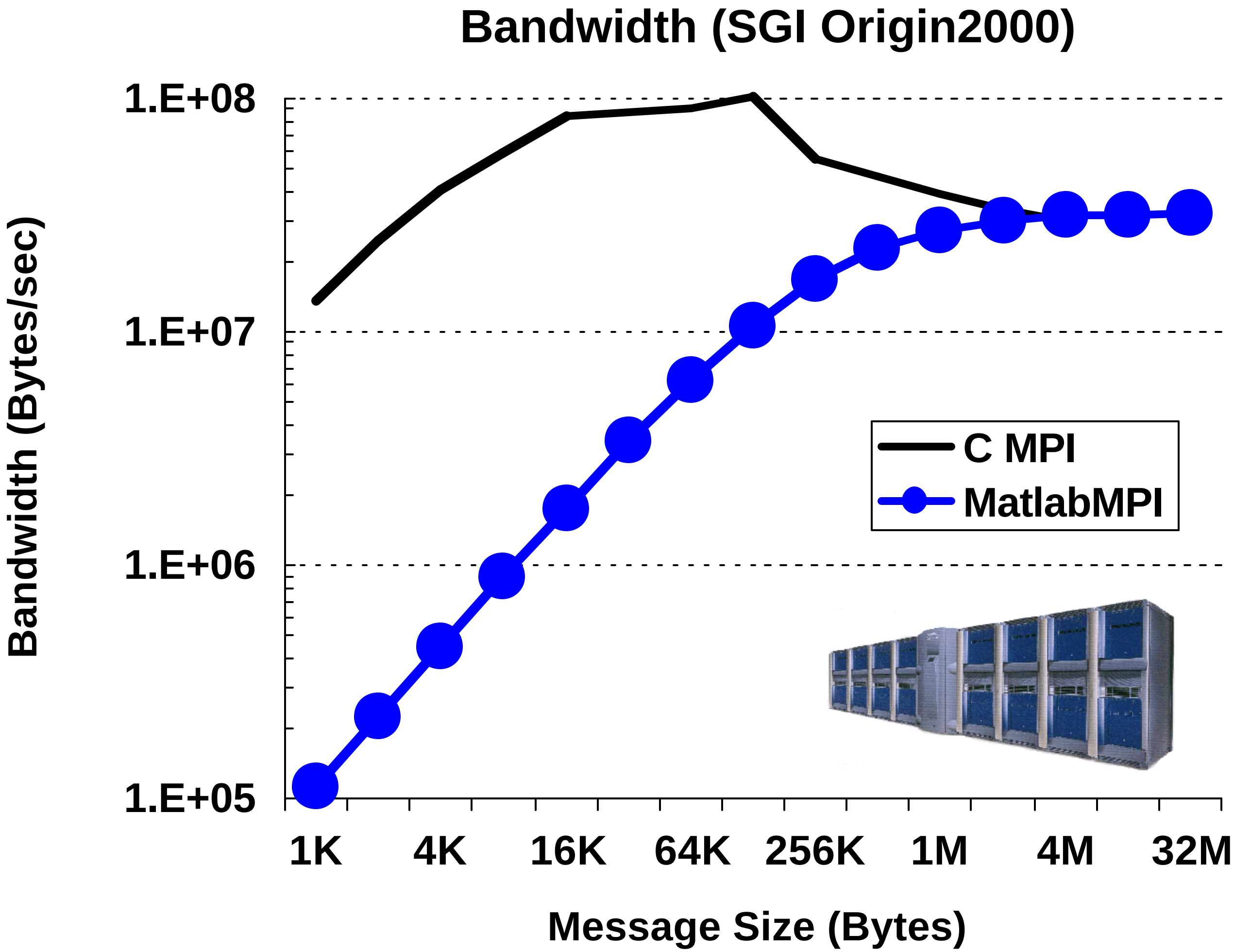
- "Core Plus" Overloads entire Matlab library
  - Supports distributed cell arrays
  - Provides best performance for every mapping

  Status: Research

Outline

- Introduction
- Approach
- **Performance Results**
  - ***MatlabMPI***
  - *pMatlab*
- Future Work and Summary

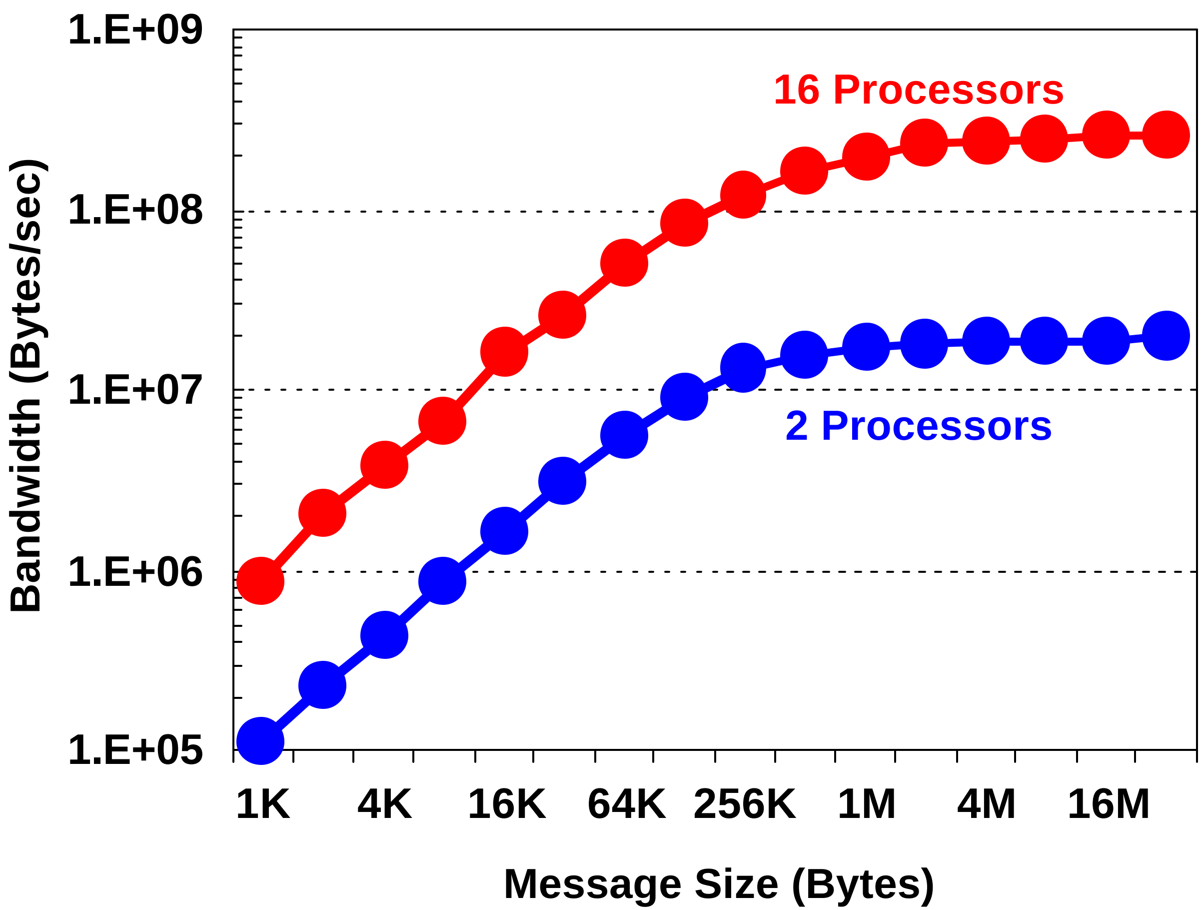
MatlabMPI vs MPI bandwidth



- Bandwidth matches native C MPI at large message size
- Primary difference is latency (35 milliseconds vs. 30 microseconds)

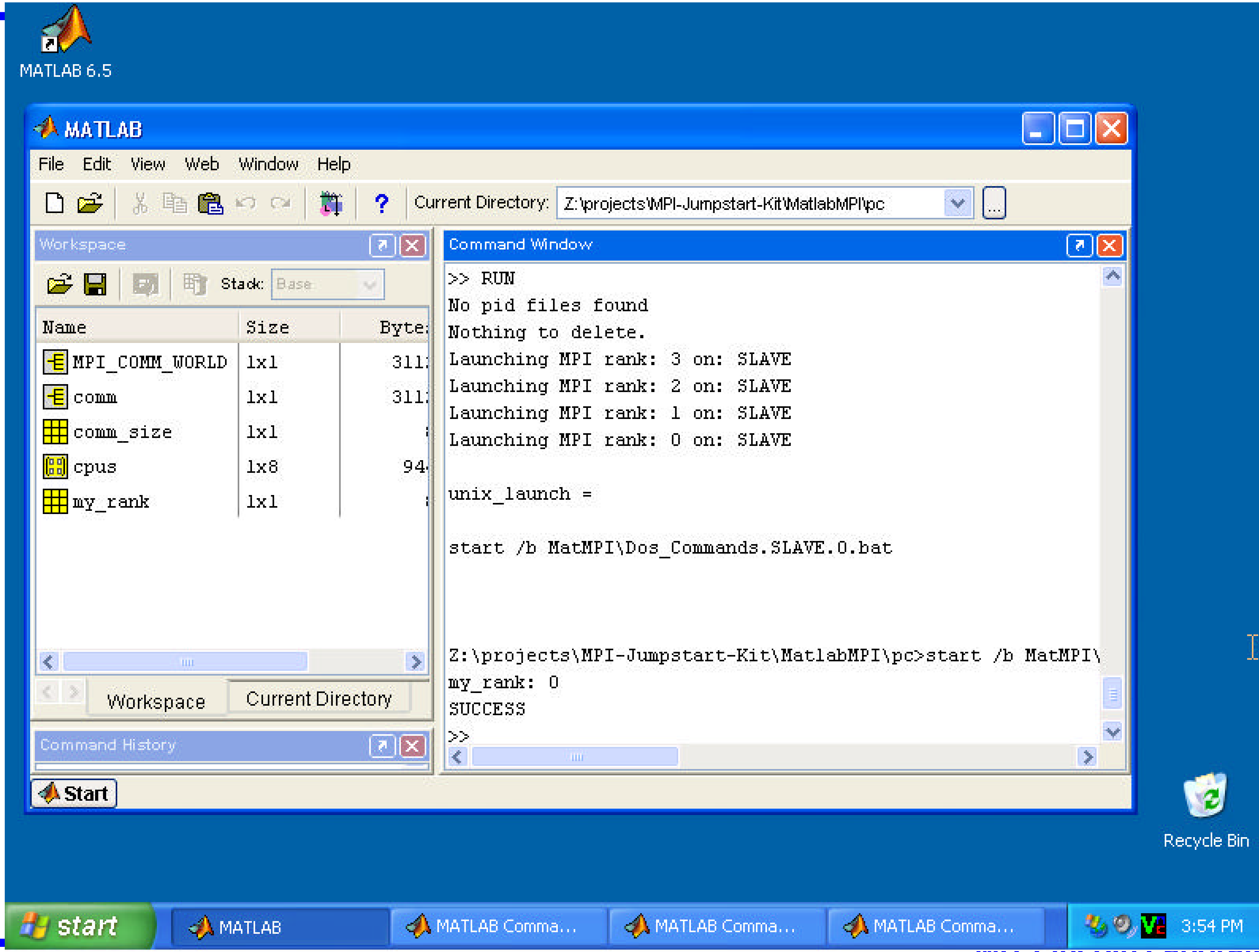
MatlabMPI bandwidth scalability

Linux w/Gigabit Ethernet

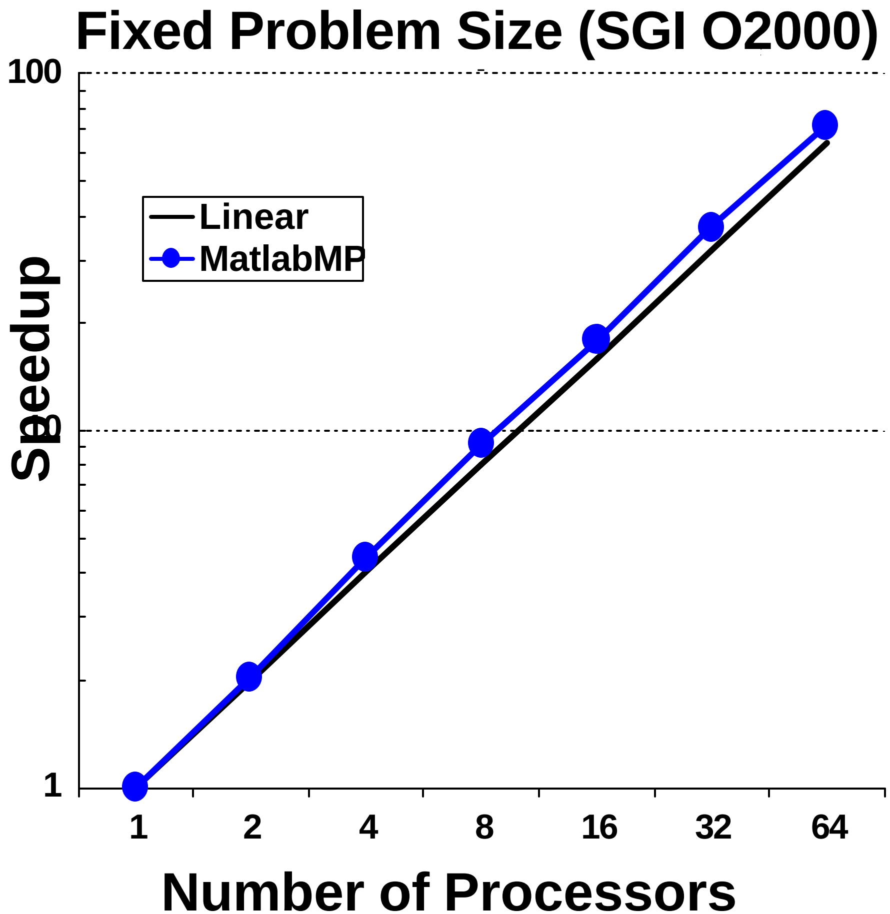


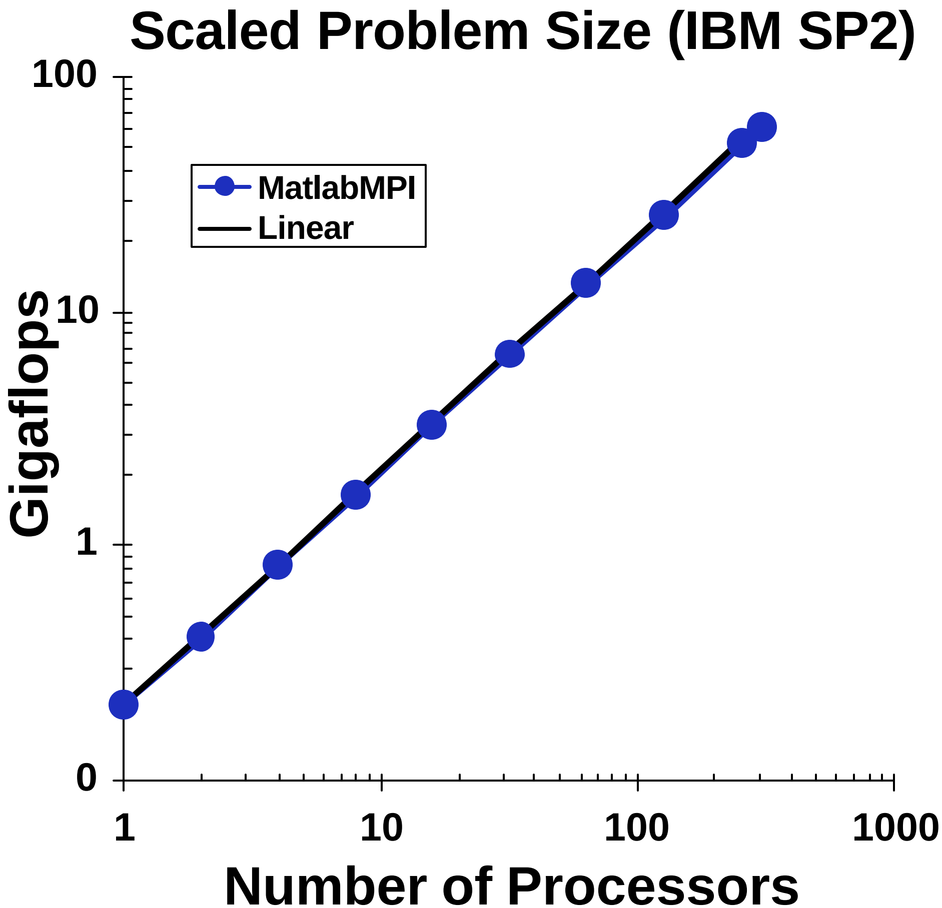
- Bandwidth scales to multiple processors
- Cross mounting eliminates bottlenecks

MatlabMPI on WindowsXP



MatlabMPI Image Filtering Performance





- Achieved “classic” super-linear speedup on fixed problem
- Achieved speedup of ~300 on 304 processors on scaled problem

“Cognitive” Algorithms

- Challenge: applications requiring vast data; real-time; large memory
- Approach: test parallel processing feasibility using MatlabMPI software
- Results: algorithms rich in parallelism; significant acceleration achieved with minimal (100x less) programmer effort

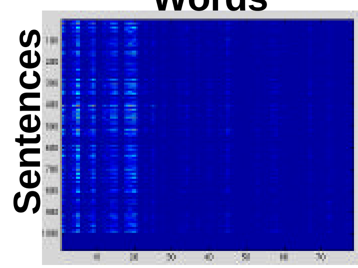
Contextual vision

Image | Face Map

Torralba (AI Lab) / Kepner (Lincoln)

Coarse Grained
Image Parallel
(Static Client Server)

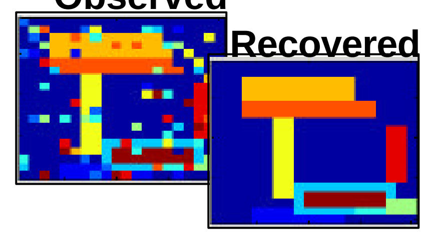
Text Processing



Murphy (AI Lab) / Kepner (Lincoln)

Medium Grained
Sentence Parallel
(Block Cyclic Dynamic Client Server)

Image Segmentation



Murphy (AI Lab) / Kepner (Lincoln)

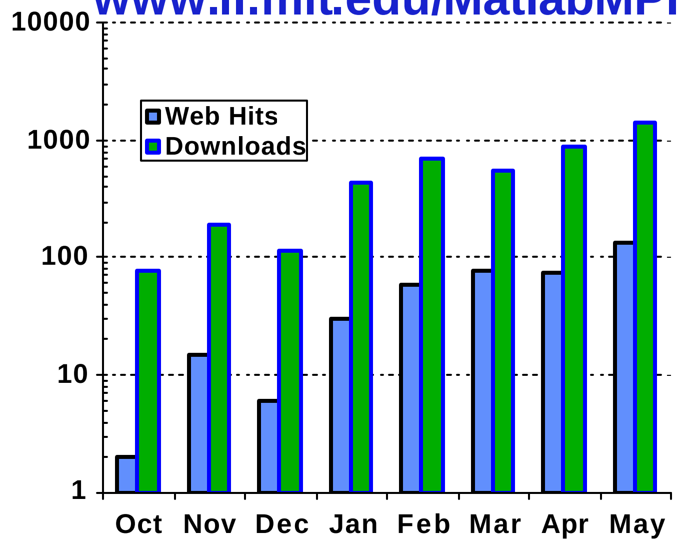
Fine Grained
Pixel Parallel
(Block Nearest Neighnor Overlap)

Application	Algorithm	CPUs / Speedup / Effort
Contextual vision	Statistical object detection	16 / 9.4x / 3 hrs
Text processing	Expectation maximization	14 / 9.7x / 8 hrs
Image segment.	Belief propagation	12 / 8x -˘ x / 4 hrs

Current MatlabMPI deployment

- Lincoln Signal processing (7.8 on 8 cpus, 9.4 on 8 duals)
- Lincoln Radar simulation (7.5 on 8 cpus, 11.5 on 8 duals)
- Lincoln Hyperspectral Imaging (~3 on 3 cpus)
- MIT LCS Beowulf (11 Gflops on 9 duals)
- MIT AI Lab Machine Vision
- OSU EM Simulations
- ARL SAR Image Enhancement
- Wash U Hearing Aid Simulations
- So. Ill. Benchmarking
- JHU Digital Beamforming
- ISL Radar simulation
- URI Heart modeling

www.ll.mit.edu/MatlabMPI

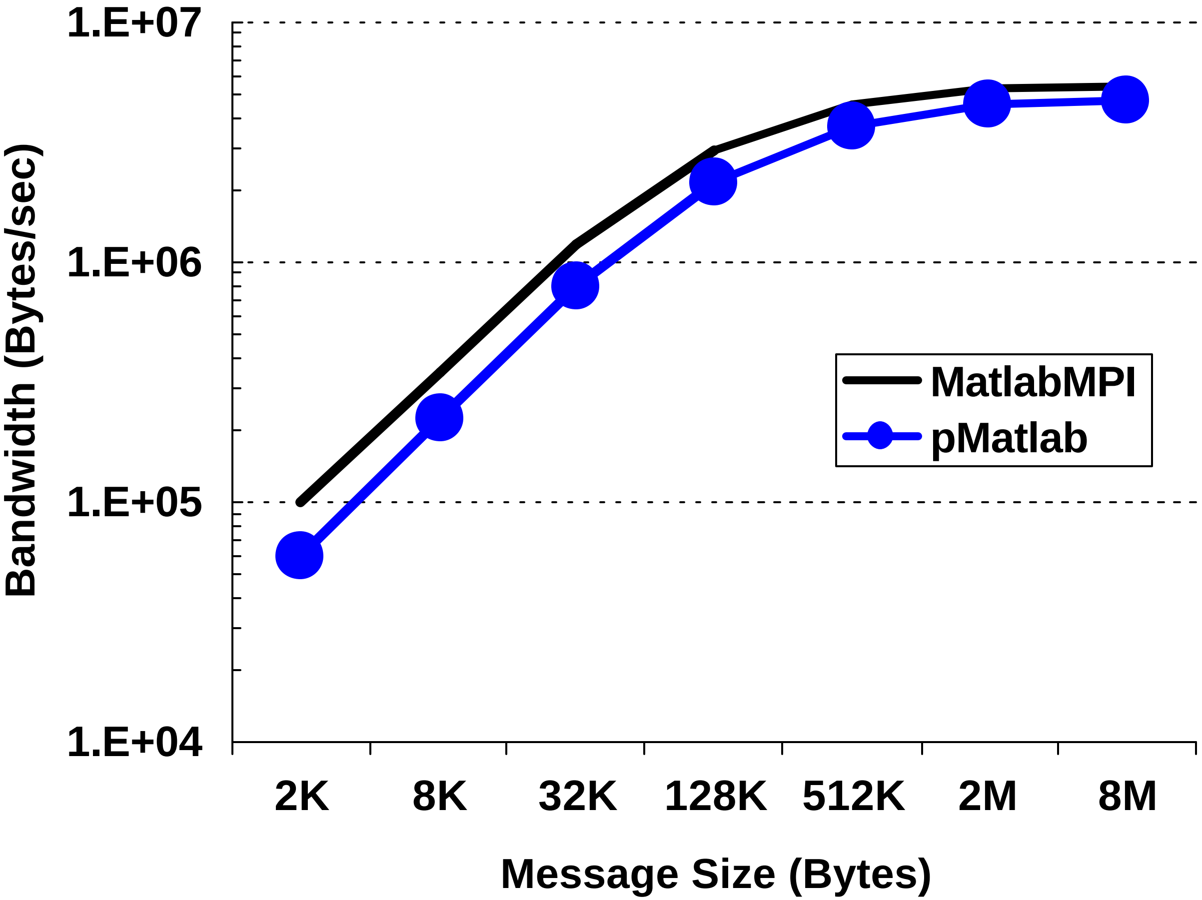


- Rapidly growing MatlabMPI user base
- Web release creating hundreds of users
  http://www.ll.mit.edu/MatlabMPI

Outline

- Introduction
- Approach
- **Performance Results**
  - *MatlabMPI*
  - ***pMatlab***
- Future Work and Summary

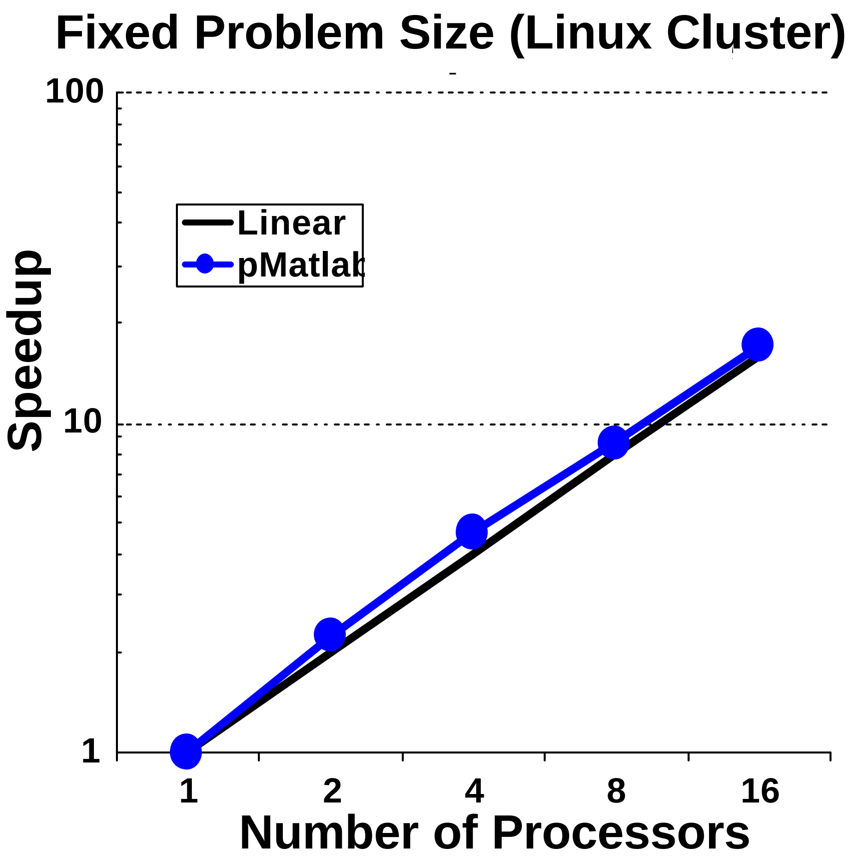
pMatlab vs. MatlabMPI bandwidth



Linux Cluster

- Bandwidth matches underlying MatlabMPI
- Primary difference is latency (35 milliseconds vs. 70 milliseconds)

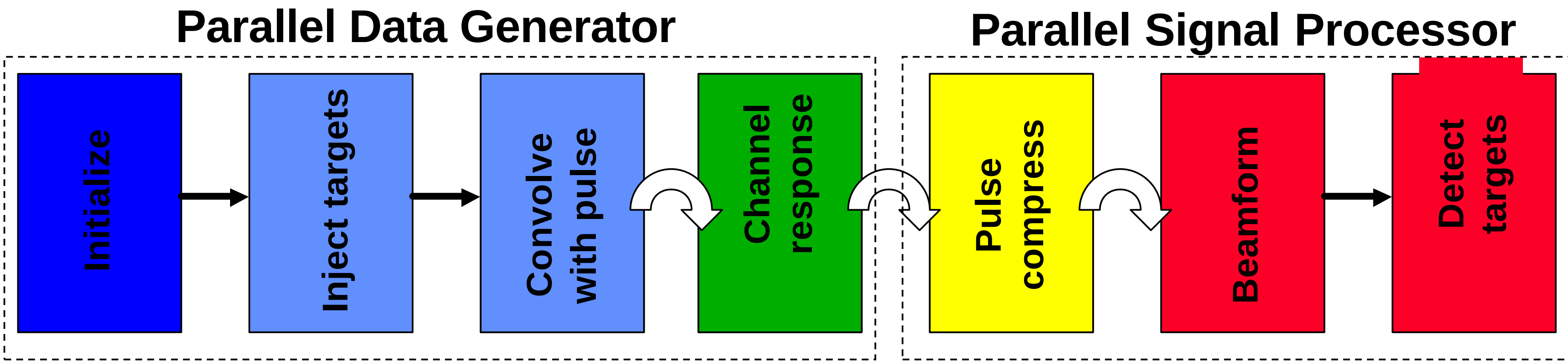
Clutter Simulation Performance




```
% Initialize
pMATLAB_Init; Ncpus=comm_vars.comm_size;

% Map X to first half and Y to second half.
mapX=map([1 Ncpus/2],{},[1:Ncpus/2])
mapY=map([Ncpus/2 1],{},[Ncpus/2+1:Ncpus]);

% Create arrays.
X = complex(rand(N,M,mapX),rand(N,M,mapX));
Y = complex(zeros(N,M,mapY);

% Initialize coefficents
coefs = ...
weights = ...

% Parallel filter + corner turn.
Y(:,:) = conv2(coefs,X);
% Parallel matrix multiply.
Y(:,:) = weights*Y;

% Finalize pMATLAB and exit.
pMATLAB_Finalize; exit;
```

- **Achieved “classic” super-linear speedup on fixed problem**
- **Serial and Parallel code “identical”**

Eight Stage Simulator Pipeline



Example Processor Distribution

- 0, 1
- 2, 3
- 4, 5
- 6, 7
- all

Matlab Map Code

```
map3 = map([2 1], {}, 0:1);
map2 = map([1 2], {}, 2:3);
map1 = map([2 1], {}, 4:5);
map0 = map([1 2], {}, 6:7);
```

- Goal: create simulated data and use to test signal processing
- parallelize all stages; requires 3 “corner turns”
- pMatlab allows serial and parallel code to be nearly identical
- Easy to change parallel mapping; set map=1 to get serial code

pMatlab Code

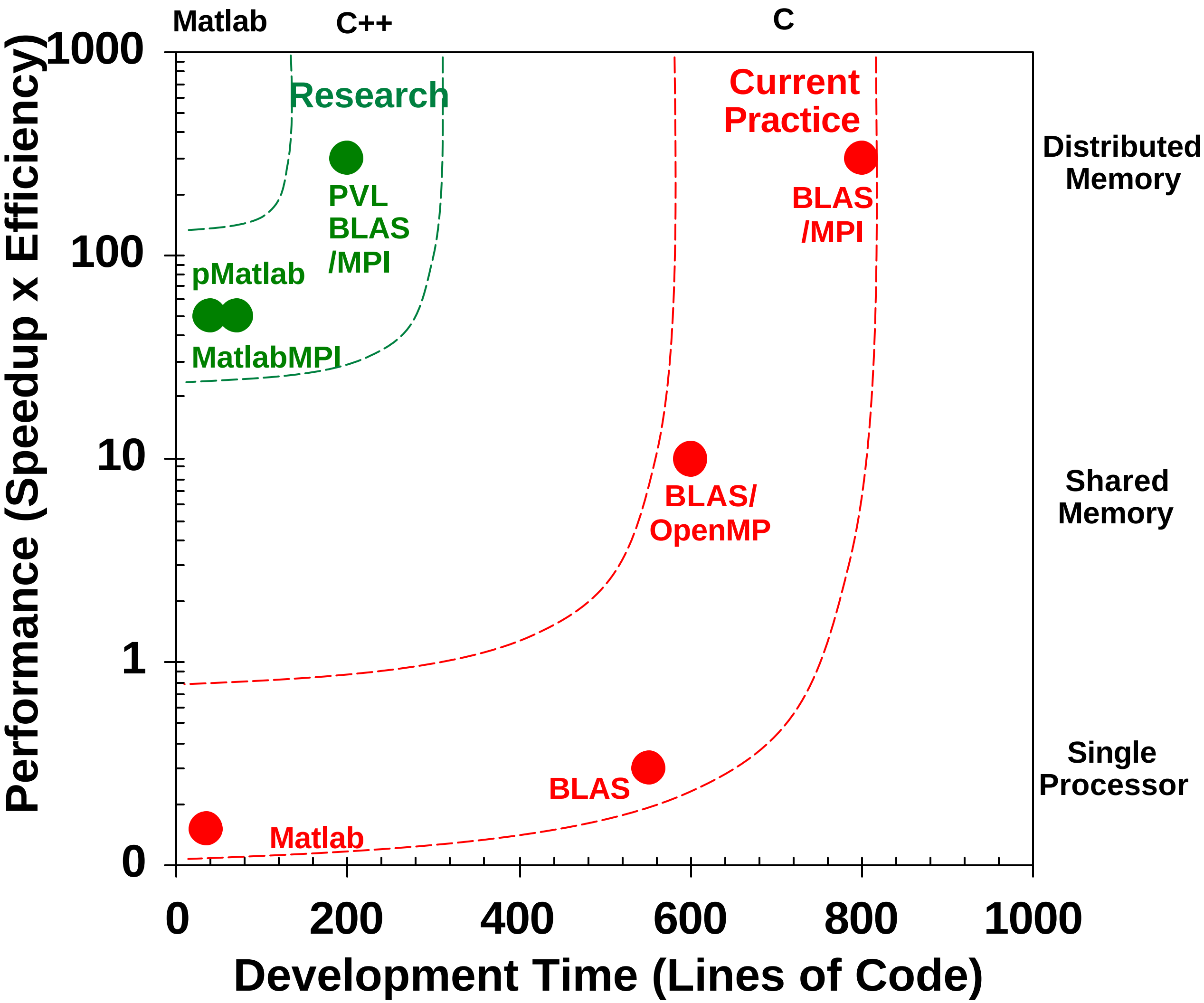
```
pMATLAB_Init; SetParameters; SetMaps;       %Initialize.
Xrand = 0.01*squeeze(complex(rand(Ns,Nb, map0),rand(Ns,Nb, map0)));
X0 = squeeze(complex(zeros(Ns,Nb, map0)));
X1 = squeeze(complex(zeros(Ns,Nb, map1)));
X2 = squeeze(complex(zeros(Ns,Nc, map2)));
X3 = squeeze(complex(zeros(Ns,Nc, map3)));
X4 = squeeze(complex(zeros(Ns,Nb, map3)));
...
for i_time=1:NUM_TIME                        % Loop over time steps.

  X0(:,:) = Xrand;                           % Initialize data
  for i_target=1:NUM_TARGETS
    [i_s i_c] = targets(i_time,i_target,:);
    X0(i_s,i_c) = 1;                         % Insert targets.
  end
  X1(:,:) = conv2(X0,pulse_shape,'same');    % Convolve and corner turn.
  X2(:,:) = X1*steering_vectors;             % Channelize and corner turn.
  X3(:,:) = conv2(X2,kernel,'same');         % Pulse compress and corner turn.
  X4(:,:) = X3*steering_vectors';            % Beamform.
  [i_range,i_beam] = find(abs(X4) > DET);    % Detect targets
end
pMATLAB_Finalize;                            % Finalize.
```

■ Implicitly Parallel Code   ■ Required Change

Outline

- Introduction
- Approach
- Performance Results
- **Future Work and Summary**

Peak Performance vs Effort

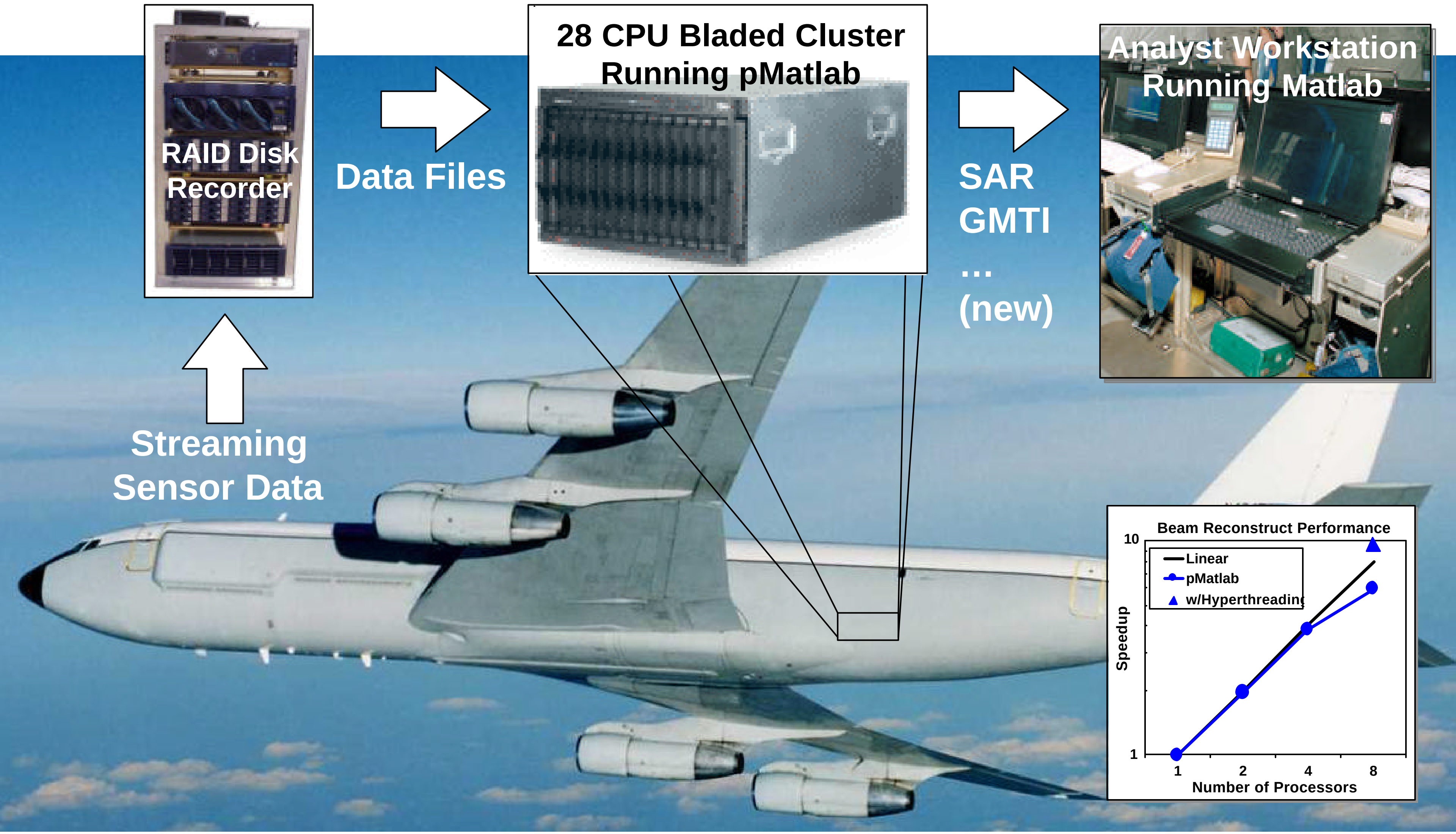


- Same application (image filtering)
- Same programmer
- Different langs/libs
  - •Matlab
  - •BLAS
  - •BLAS/OpenMP
  - •BLAS/MPI*
  - •PVL/BLAS/MPI*
  - •MatlabMPI
  - •pMatlab*

*Estimate

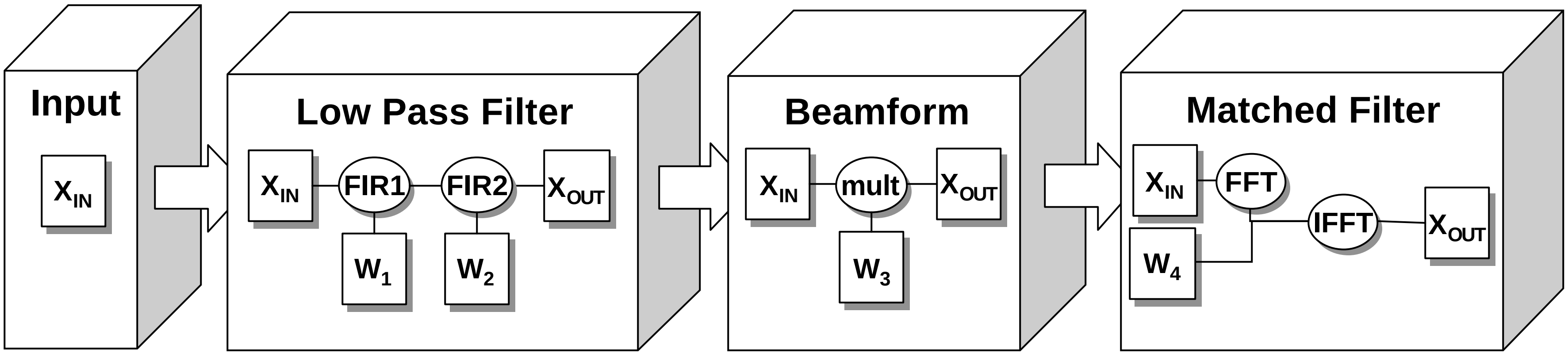
pMatlab achieves high performance with very little effort

Airborne Sensor “QuickLook” Capability



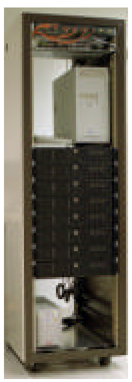
pMatlab Future Work

1. Demonstrate in a large multi-stage framework



2. Incorporate Expert Knowledge into Standard Components

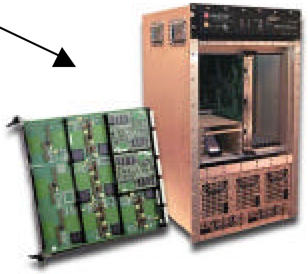
User Workstation



Lincoln GRID

Special Cluster

Embedded Board



Embedded Multi-computer

3. Port pMatlab to HPEC systems

Summary

- MatlabMPI has the basic functions necessary for parallel programming
  - Size, rank, send, receive, launch
  - Enables complex applications or libraries
- Performance can match native MPI at large message sizes
- Demonstrated scaling into hundreds of processors
- pMatlab allows user's to write very complex parallel codes
  - Built on top of MatlabMPI
  - Pure Matlab (runs everywhere Matlab runs)
  - Performace comparable to MatlabMPI
- Working with MIT LCS, Ohio St. and UCSB to define a unified parallel Matlab interface

Acknowledgements

- **Support**
  - Charlie Holland DUSD(S&T) and John Grosh OSD
  - Bob Bond and Ken Senne (Lincoln)
- **Collaborators**
  - Nadya Travinin (Lincoln)
  - Stan Ahalt and John Nehrbass (Ohio St.)
  - Alan Edelman and Ron Choy (MIT LCS)
  - John Gilbert (UCSB)
  - Antonio Torralba and Kevin Murphy (MIT AI Lab)
- **Centers**
  - Maui High Performance Computing Center
  - Boston University
  - MIT Earth and Atmospheric Sciences

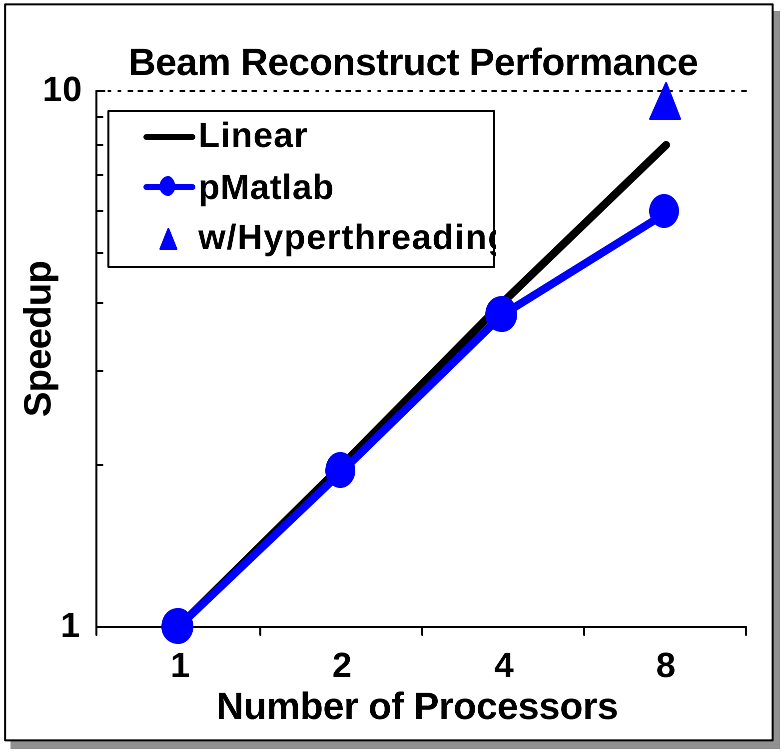
Web Links

MatlabMPI

http://www.ll.mit.edu/MatlabMPI

High Performance Embedded Computing Workshop

http://www.ll.mit.edu/HPEC


Beam Reconstruct Performance
10
1
Linear
pMatlab
w/Hyperthreading
Speedup
1
2
4
8
Number of Processors